Andreas Kevin (17/415896/PA/18165)
Ghinaya Fairuz Zahra (16/392767/PA/17071)

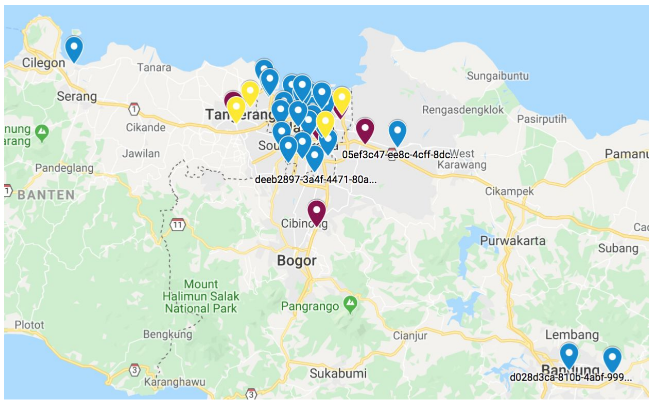
Final Project Science Management

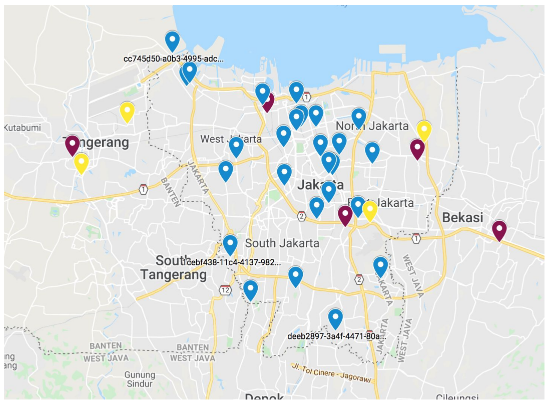
1. **Problem:**
   Find the shortest and most efficient path taken when a customer wants to go to a Hub and Hubs to Warehouses.

2. **Dataset:**
   The dataset given by BliBli contained:
   - CustomerID
   - CustomerLatitude
   - CustomerLangitude
   - HubID
   - HubLatitude
   - HubLongitude
   - WarehouseId
   - WarehouseLatitude
   - WarehouseLongitude

   From this data, we are able to know the unique ID and find the exact location of the customers, hubs, and warehouses.

   The first picture is the full map of BliBli warehouses, customers and Hubs in Indonesia and the second picture is focused on the Jabodetabek area.





Yellow: Warehouse
Red: Hubs
Blue: Customers

3. **Technical Steps:**

Github Link: http://bit.do/SMTeam5

a. The programming language used: Python
b. The libraries used:
   i. pandas
   ii. numpy
   iii. math
c. Importing excel files into python. Noting that longitude is ‘x’ and latitude is ‘y’.
d. Using the Euclidean metric to find the "ordinary" straight-line distance between two points in Euclidean space. Mathematically, it's calculated using Pythagoras' theorem. The square of the total distance between two objects is the sum of the squares of the distances along each perpendicular co-ordinate. In this case, we have to find the distance of:
   i. Customers to Hub
   ii. Hub to Warehouse
   iii. Hub to Hub
e. Actually using the euclidean distance is not the best way to find the distance in this case as we use latitude and longitude. However for the sake of simplicity we will use it
f. For this case, we will be using the Dijkstra algorithm using a double-ended queue. It is where items can be added or removed from either end without any type of restrictions.
g. First is imports and data formats using namedtuple for storing. Adding a default value to the cost argument. It is also used to initialize data

```
def make_edge(start, end, cost=1):
  return Edge(start, end, cost)
class Graph:
    def __init__(self, edges):
        wrong_edges = [i for i in edges if len(i) not in [2, 3]]
        if wrong_edges:
            raise ValueError('Wrong edges data: {}'.format(wrong_edges))

        self.edges = [make_edge(*edge) for edge in edges]
```

h. Next is to define the vertices. From the starting to the ending node.

```
@property
    def vertices(self):
        return set(
            sum(
                ([edge.start, edge.end] for edge in self.edges), []
            )
        )
```

i. Next is to add and remove functionality (nodes and edges)

```
def get_node_pairs(self, n1, n2, both_ends=True):
        if both_ends:
            node_pairs = [[n1, n2], [n2, n1]]
        else:
            node_pairs = [[n1, n2]]
        return node_pairs

    def remove_edge(self, n1, n2, both_ends=True):
        node_pairs = self.get_node_pairs(n1, n2, both_ends)
        edges = self.edges[:]
        for edge in edges:
            if [edge.start, edge.end] in node_pairs:
                self.edges.remove(edge)

    def add_edge(self, n1, n2, cost=1, both_ends=True):
        node_pairs = self.get_node_pairs(n1, n2, both_ends)
        for edge in self.edges:
            if [edge.start, edge.end] in node_pairs:
                return ValueError('Edge {} {} already exists'.format(n1, n2))
```

j. Detecting node neighbors to see which nodes are closest

```
 @property
    def neighbours(self):
        neighbours = {vertex: set() for vertex in self.vertices}
        for edge in self.edges:
            neighbours[edge.start].add((edge.end, edge.cost))

        return neighbours
```

k. Now we implement the actual Dijkstra algorithm. IT finds the shortest path between nodes in the graph but in this version, we use a source node to find the shortest path from the source node to the destination node. In this case the source node is Hub and the destination node is Warehouse.

```
def dijkstra(self, source, dest):
        assert source in self.vertices, 'Such source node doesn\'t exist'
```

```
distances = {vertex: inf for vertex in self.vertices}
previous_vertices = {
    vertex: None for vertex in self.vertices
}
distances[source] = 0
vertices = self.vertices.copy()


while vertices:
    current_vertex = min(
        vertices, key=lambda vertex: distances[vertex])
    vertices.remove(current_vertex)
    if distances[current_vertex] == inf:
        break
    for neighbour, cost in self.neighbours[current_vertex]:
        alternative_route = distances[current_vertex] + cost
        if alternative_route < distances[neighbour]:
            distances[neighbour] = alternative_route
            previous_vertices[neighbour] = current_vertex

path, current_vertex = deque(), dest
while previous_vertices[current_vertex] is not None:
    path.appendleft(current_vertex)
    current_vertex = previous_vertices[current_vertex]
if path:
    path.appendleft(current_vertex)

distance_between_nodes = 0
for index in range(1, len(path)):
    for thing in self.edges:
        if thing.start == path[index - 1] and thing.end ==
path[index]:
            distance_between_nodes += thing.cost
path2 = list(path)
return path2, distance_between_nodes
```

l. Now we make a class for each instance which are Customer, Warehouse and Hub. This is used to identify itself
m. From grapharray, we define the distance from:
    i. Hubs to Hubs
    ii. Hubs to Warehouse
n. This is used to create a list of all the connections and then we will sort it accordingly

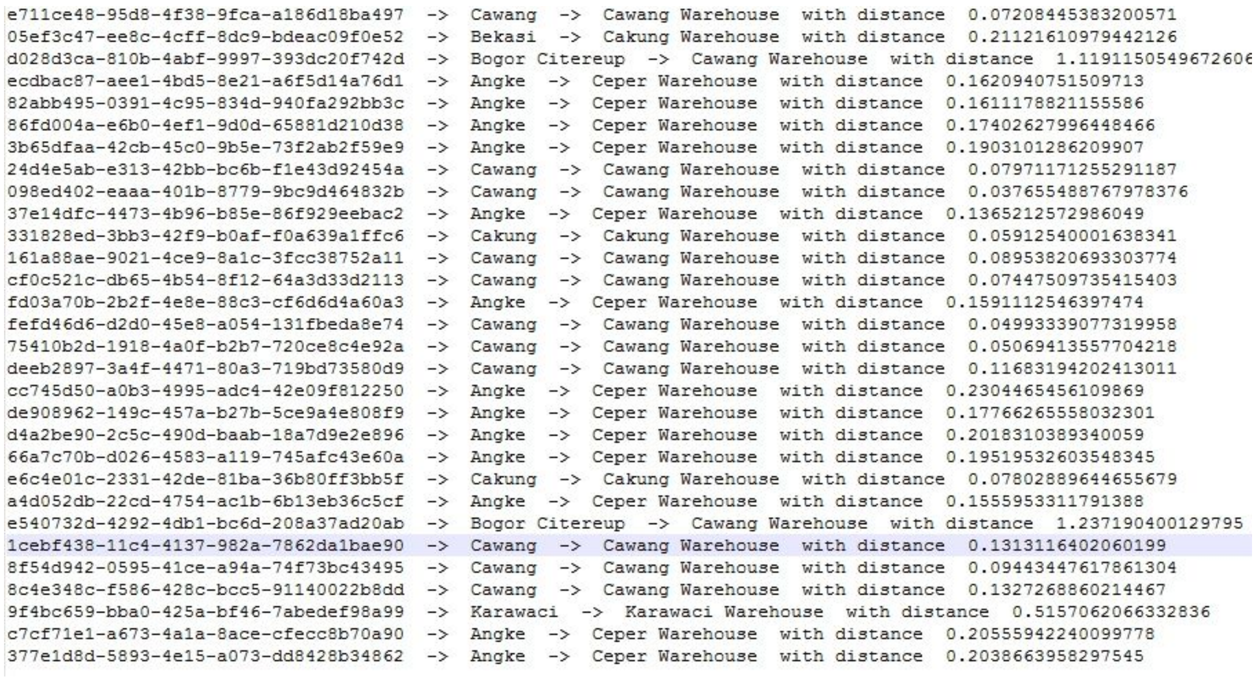
```
graph = Graph(grapharray)
grapharr = []
for i in range(0,len(hubID)):
    for j in range(0,len(warehouseID)):
        defGraph = graph.dijkstra(hubsList[i].id,warehouseList[j].id)
        grapharr.append(defGraph
```

o. After graph is sorted we will print it. Then as customers cannot go to warehouses directly, we will define the distance between every customer to every hub available.

```
for i in range (0, len(customerID)):
    print(customersList[i].id, " -> ", customersList[i].nearestHub()[1], "
-> ", end = ' ')
    dist.append(customersList[i].nearestHub()[0])
```

p. For every hub and warehouse nodes (point location) in the graph. The graph will be able to show the connection or distance between the Hub and the Warehouse. It only takes the shortest path possible between these 2 nodes.
q. Then we calculate the different distances between the hubs and warehouse and print accordingly.
r. The output will show the shortest distance from Hub to Warehouse

4. **Results:**



```
e711ce48-95d8-4f38-9fca-a186d18ba497  ->  Cawang  ->  Cawang Warehouse  with distance  0.07208445383200571
05ef3c47-ee8c-4cff-8dc9-bdeac09f0e52  ->  Bekasi  ->  Cakung Warehouse  with distance  0.21121610979442126
d028d3ca-810b-4abf-9997-393dc20f742d  ->  Bogor Citereup  ->  Cawang Warehouse  with distance  1.119115054967260
ecdbac87-aee1-4bd5-8e21-a6f5d14a76d1  ->  Angke  ->  Ceper Warehouse  with distance  0.1620940751509713
82abb495-0391-4c95-834d-940fa292bb3c  ->  Angke  ->  Ceper Warehouse  with distance  0.1611178821155586
86fd004a-e6b0-4ef1-9d0d-65881d210d38  ->  Angke  ->  Ceper Warehouse  with distance  0.17402627996448466
3b65dfaa-42cb-45c0-9b5e-73f2ab2f59e9  ->  Angke  ->  Ceper Warehouse  with distance  0.1903101286209907
24d4e5ab-e313-42bb-bc6b-f1e43d92454a  ->  Cawang  ->  Cawang Warehouse  with distance  0.07971171255291187
098ed402-eaaa-401b-8779-9bc9d464832b  ->  Cawang  ->  Cawang Warehouse  with distance  0.037655488767978376
37e14dfc-4473-4b96-b85e-86f929eebac2  ->  Angke  ->  Ceper Warehouse  with distance  0.1365212572986049
331828ed-3bb3-42f9-b0af-f0a639a1ffc6  ->  Cakung  ->  Cakung Warehouse  with distance  0.05912540001638341
161a88ae-9021-4ce9-8a1c-3fcc38752a11  ->  Cawang  ->  Cawang Warehouse  with distance  0.08953820693303774
cf0c521c-db65-4b54-8f12-64a3d33d2113  ->  Cawang  ->  Cawang Warehouse  with distance  0.07447509735415403
fd03a70b-2b2f-4e8e-88c3-cf6d6d4a60a3  ->  Angke  ->  Ceper Warehouse  with distance  0.1591112546397474
fefd46d6-d2d0-45e8-a054-131fbeda8e74  ->  Cawang  ->  Cawang Warehouse  with distance  0.04993339077319958
75410b2d-1918-4a0f-b2b7-720ce8c4e92a  ->  Cawang  ->  Cawang Warehouse  with distance  0.05069413557704218
deeb2897-3a4f-4471-80a3-719bd73580d9  ->  Cawang  ->  Cawang Warehouse  with distance  0.11683194202413011
cc745d50-a0b3-4995-adc4-42e09f812250  ->  Angke  ->  Ceper Warehouse  with distance  0.2304465456109869
de908962-149c-457a-b27b-5ce9a4e808f9  ->  Angke  ->  Ceper Warehouse  with distance  0.17766265558032301
d4a2be90-2c5c-490d-baab-18a7d9e2e896  ->  Angke  ->  Ceper Warehouse  with distance  0.2018310389340059
66a7c70b-d026-4583-a119-745afc43e60a  ->  Angke  ->  Ceper Warehouse  with distance  0.19519532603548345
e6c4e01c-2331-42de-81ba-36b80ff3bb5f  ->  Cakung  ->  Cakung Warehouse  with distance  0.07802889644655679
a4d052db-22cd-4754-ac1b-6b13eb36c5cf  ->  Angke  ->  Ceper Warehouse  with distance  0.1555953311791388
e540732d-4292-4db1-bc6d-208a37ad20ab  ->  Bogor Citereup  ->  Cawang Warehouse  with distance  1.237190400129795
1cebf438-11c4-4137-982a-7862da1bae90  ->  Cawang  ->  Cawang Warehouse  with distance  0.1313116402060199
8f54d942-0595-41ce-a94a-74f73bc43495  ->  Cawang  ->  Cawang Warehouse  with distance  0.09443447617861304
8c4e348c-f586-428c-bcc5-91140022b8dd  ->  Cawang  ->  Cawang Warehouse  with distance  0.1327268860214467
9f4bc659-bba0-425a-bf46-7abedef98a99  ->  Karawaci  ->  Karawaci Warehouse  with distance  0.5157062066332836
c7cf71e1-a673-4a1a-8ace-cfecc8b70a90  ->  Angke  ->  Ceper Warehouse  with distance  0.20555942240099778
377e1d8d-5893-4e15-a073-dd8428b34862  ->  Angke  ->  Ceper Warehouse  with distance  0.2038663958297545
```